

The Sourceror's Apprentice

The Assembly Language Journal of Merlin Programmers

Vol 1 No 11 Nov-Dec 1989

Out With the Old, In With the New

The Apple II had a tough time in the closing moments of 1989. Apple's pre-Christmas profits were far less than expected (with the rumorists and *USA TODAY* blaming the Apple II's slow sales!), and our beloved *CALL A.P.P.L.E.* expired. The next issue we receive will no doubt be the last.

Before you all run out and buy a NeXT (ha!), stop and consider two things:

First, I had an inkling about all this back in August. To wit, "...Apple has yet to reap all the consequences of years of neglect and exploitation. Computer markets turn slowly nowadays, and Apple's moderate amount of support at present will not stem the tide in the short term." (Vol.1 No. 7, p.3)

Second, blame for the demise of *CALL A.P.P.L.E.*, the only multi-language technical journal for the II, can be at least partly laid at the feet of Apple, Inc. Listen carefully, now, I am not denigrating those who work on and champion the Apple II at Apple, Inc. Those folks are doing a tremendous work. I *am* suggesting, however, that one of the unfortunate ramifications of Apple's decision to take APDA back in-house was that the move left Tech Alliance all dressed up with no place to go. The co-op had hired lots of employees and managers, made capital investments in hardware and buildings, and had built APDA into a fairly well-established concern. With the rug pulled out from under them, they were left scrambling to cut costs as fast as possible. I can assure you that is harder to successfully pull off than expansion.

My hunch is that, for whatever reasons (the rumor mill has churned out a hundred stories) the *CALL A.P.P.L.E.* folks couldn't make it happen in time to keep the magazine from financial disaster.

In conclusion, friends, the passing of our beloved *CALL A.P.P.L.E.* is *not* really a commentary on the state of the Apple II market. It is the unfortunate symptom of a long chain of events. I don't know if Apple, Inc.'s decision to take back APDA was good or evil - but it had a nasty side effect. Though it be a grievous wind that hath blown in our faces this holiday season, there is yet reason for hope - Apple obviously *does* have plans for the II line, and my sources suggest some actual marketing money in

the pipeline.

As I've said before, I think there shall be rewards for those who persevere, but even now the worst is probably not over.

On the positive side, one idea I've heard floated is for Apple to appoint an "Apple II Czar", i.e. someone to put the corporate infrastructure aright as far as the II is concerned. This has some potential, I think, especially since the stockholders really took a hit in the wallet the last few days. Stockholders don't care *which* product makes them bucks - as long as they are making their money. Bucks is bucks, after all.

The subscriber survey...

My thanks to all who took the time to return the subscriber survey. I really learned a lot. Here's how it turned out...

As of this date (early December), 80 of you responded. Not everyone answered every question, so the number of responses per question doesn't always add up to 80.

1) I find the content of the Apprentice:

- 10 said too difficult
- 12 said too simplistic
- 53 said about right

2) I find the tone of the newsletter:

- 27 said too light, cut the chatter
- 53 said about right

3) I find the page layout in this issue:

- 0 said too squished
- 43 said not enough content
- 36 said okay, a decent tradeoff

4) I find the current mix between 8 & 16 bit:

- 17 said too biased in favor of the GS
- 17 said too biased in favor of the 8 bit Apples
- 45 said about right

5) If Apple discontinues the Apple II, I would:

- 28 said buy an IBM PC or compatible
- 16 said buy a Macintosh
- 33 said "other"

6) I use my Apple II...

- 73 at home for word processing, etc.
- 37 for business purposes
- 35 for educational software

7) Topics I'd like to see...

Too numerous to even summarize - but I have lots more ideas now, thank you. We're addressing some of them this very month.

8) If the subscription price were raised to \$35 per year for 12 pages per month, I would:

- 54 said continue subscribing
- 22 said not renew

Over half (44) attached extra pages or wrote on the back. I read every letter and note.

- To all of you who asked how I can continue producing this newsletter if my margin is so small: I can afford to continue because A) I consider *The Sourceror's Apprentice* a long term investment, and B) I do a considerable amount of contract programming, consulting, and custom applications development. It was obvious, I hope, that SApp (as I call it) is not my main gig. As uncomfortable as this might make some of you feel, I recently co-authored a Macintosh product that is doing pretty well. I encourage you to delight in the ironic fact that, for once, something on the Macintosh is subsidizing something on the Apple II. I know I do.

- To those who wondered if I weren't "too nice" to be in business - now there is a criticism I can take! In actuality, I cannot figure out why business people in general are not the nicest human beings on the planet - after all, they're trying to persuade you to voluntarily give them your money. I, for one, don't do business with anyone who isn't trying very hard to keep me happy. As you'll see in a few paragraphs, the subscriber survey has convinced me that I have not been being nice enough! (Although it is a fact that surveys of this kind tend to get the most satisfied and the least satisfied to respond.)

The way we were...

I have not been able to offer nearly as much of my time to *The Apprentice* as I would have liked. Robert Muir (the letter I lead with last month) was right about that. That's part of the reason why the tone of this rag has been pretty informal and the distribution schedule pretty loose. As I mentioned above, that's also why we've been able to continue when others have croaked.

Still, I don't really think there is anyone in a better position to publish something of this nature (a conceit, perhaps, but we entrepreneurs have *got* to

believe in ourselves), and I also don't think it can be produced any cheaper. Everyone wants more for their money (see survey question #3!), but as I explained last month, it can't be done with our present structure.

It is a foolish businessman, however, who doesn't listen to his customers. Well over half of you who responded want more for your money, and it now behooves me to figure out a way to make it happen.

...and the way we shall be.

- The bad news first: a small price increase. One year will now be \$29.95, two years will \$56. The quarterly disk will be \$25 per year.

In return, I am "professionalizing" this publication somewhat. I am hiring out the disk duplication duties so that they can be distributed in a more timely fashion, and I am negotiating with one person to be an associate editor and another high powered type to be a regular columnist. They are both *good* and would really help bump us up to the next notch in the publishing hierarchy.

- To offset their pay and to provide funds for an expanded format, I am going to aggressively pursue advertisers. With *CALL A.P.P.L.E.* out of the picture, we are now one the primary contact points with the Apple II programming community. If you have developed something for programmers or have hardware for sale, please consider an ad here.

Don't expect a glossy cover and four color ads. But we are most definitely going to do our level best to be responsive to your desires. Incidentally, our ad policy will not allow the sacrifice of editorial space for advertising space. My intention is to use the ad monies (when and if we can get them) to finance additional articles.

CALL A.R.T.I.C.L.E.S

Since we're going to be needing more quality code and articles than I and my cohorts could possibly generate, I am hereby requesting that the 12 of you who found this newsletter too simplistic start writing for us (and contact your hotdog buddies, too). I've moved the pay up a notch, we're looking at \$75 - \$125 for a nice piece that requires neither too much rewriting or recoding on my part. All submissions require articles in unformatted text files and source code in Merlin format.

As for the other survey questions... I thought it hilarious that there was an exact tie betwixt those who want more GS stuff and those who don't. It's a no-win deal for an Apple II publisher. I've even

heard the boys at A2-Central moaning about this.

I also found it interesting that our survey yielded only 16 of 80 who would move to the Mac if Apple ended the life of the II. You can bet that I'll be forwarding the results to Mr. Sculley. The "other" computer of choice was probably the Amiga.

All in all, the survey results were most encouraging. My thanks to everyone, and especially to those who took the time to share their ideas, insights, and kind words. You Apple II folks are an intelligent, articulate bunch, not to mention patient and kind (well, most of you, anyway).

A GS BASIC 4U?

Micol Systems, Canada, is up to version 3.5 of their GS BASIC. Up until now I've been fairly lukewarm about the product. It has some nice features, but Micol was making some decisions I really couldn't understand, including only supporting the linking of assembly files generated by their own assembler.

You can guess how I felt about that, being one of the world's foremost Merlin promoters.

After a long period of discussion, the Micol gang has finally come around to my way of thinking. We are currently E-MAILing each other silly trying to work out the details. I plan an article or series of articles on mixing Micol with Merlin.

The Micol people are also planning some other very intelligent moves, so I am therefore finally offering the software for sale for \$95 to subscribers (shipping not included). The suggested retail is \$149.95.

Incidentally, in a no-holds-barred effort to get back on our publishing schedule, this issue is a doubler, meaning it includes the material for both November and December. I know I had a lot of non-programming material to discuss, but at least it was two months worth!

We aim to please, though, and if the idea of a "one fer two" bothers you, drop us postcard and we'll extend your subscription a month.

I hope you had a blessed Christmas and I give you all my best wishes for a happy New Year - and New Decade, too.

== Ross ==

Jumping Around, Hiring a Picker, & a P8 MLI Error Handler

by Ross W. Lambert, Editor

One of the most popular types of articles requested in the subscriber survey was that of pre-cooked and reusable subroutines. It reminds me of my days as a teacher - whenever a specialist would come to "inservice" us poor schmucks, we'd invariably cry, "Gimme a worksheet!", meaning "Give me something I can use right now in my classroom." They seldom did, by the way. I'll try to respond better.

In this month's listing, I have tried to give you a reusable ProDOS 8 MLI error handler that you can just link into your own code with very little modification. Not only that, but I have also attempted to illustrate a few techniques for selecting myriads of options that I have found useful.

The first section of code begins by setting up the screen. I don't care if the screen is in 40 or 80 columns - the error messages all fit correctly either way. The Imprint subroutine called in line 37 was first run in the very first *Apprentice* (Vol.1 No. 1, January, 1989). I made a minor modification for this article so I have reprinted it again. You can see it's usefulness in lines 38-41; the screen layout is done very much like you would in BASIC or another higher level language.

The Imprint routine also makes use of a 65XXX series habit of depositing the return address after a JSR right on top of the stack. In this case Don Lancaster (the original author) bumped the return address by the length of the strings to be printed so that program control would resume immediately after the embedded ASCII text. It's a neat trick, I think.

Although none of the routines in this program need parameters, a similar technique can allow us to pass data back and forth between generic routines (I'll detail this more next month). This allows for incredibly

modular programming; which is in turn the secret to productivity. I can assure you that, for many employers, the speed with which you churn out a working application is sometimes of the highest importance. Please make a mental note, however, that in certain situations where blinding speed is required, a custom in-line routine can execute faster than a generic subroutine.

Speaking of modularity, I have setup this program into three separate, independent, linkable modules. The demo module (Listing 1) is only useful to show off the other two, of course, but the embedded string printer and the MLI error handling module are ready to be linked into your own code as-is. Don't forget to declare their entry points as labels EXTERNAL to your source file.

Meanwhile, back at the BRANCH (hehehe), the demo loop in lines 43 -62 merely grabs an MLI error code from a table and passes it to the error handler. The error handler looks for a match in its own table of error numbers, jumps to the appropriate routine, displays an error message and waits for a keypress. Try to not to get excited when the demo tells you your volume bitmap may be damaged; it's only a test of the system. If this were an actual emergency...

Listing 1 - The Demo Module

```
1 *****
2 *
3 * A General Purpose P8 MLI Error Handler
4 * By Ross W. Lambert
5 *
6 * Copyright (C) 1989
7 * Ariel Publishing, Inc.
8 * All Rights Reserved
9 *
10 *****
11
12
13 * Stuff for Merlin
14
15         mx      ⌘11
16         REL
17         DSK      DemoModule.L
18
19         LST      OFF
20
21 * A few equates
22
23 OurPtr   =      $06
24
25 BELL     =      $FF3A
26 HOME     =      $FC58
27 ProDOS   =      $BF00
28 COUT     =      $FDED
29 CROUT    =      $FD8E
30 Keyboard =      $C000
31 ClrStrobe = $C010
32 PRHEX    =      $FDE3
33
34 * Declare our external references...
35
36         EXT      Imprint,errorlist,MLI_Error
37
38 * Real stuff starts here...
39
40 Start    JSR      Home
41          JSR      Imprint
42          ASC      "P8 MLI Error Trapper Demo",8D,8D
43          ASC      " Cycling through MLI errors ",8D
```

```

44      ASC      "(ESCape to quit)",8D
45      ASC      "_____",8D,00
46
47 * We'll cycle through all 30 MLI errors and display error msgs.
48
49      LDX      #28                      ;count 29 to 0 backwards
50
51 :loop   STX      ErrCount
52      LDA      errorlist,X              ;get an error
53
54      JSR      MLI_Error                ;go handle it
55                                          ;key pressed is returned in accumulator
56      CMP      #155                    ;user want to escape?
57      BNE      :cont
58      JMP      Quit                    ;yep, so leave
59
60 :cont   LDX      ErrCount
61      DEX
62      BNE      :loop                    ;just quit when done
63
64 Quit    LDA      #4
65      STA      ParmTbl
66
67      JSR      ProDOS
68      DFB      $65                      ;QUIT call to MLI
69      DA      ParmTbl
70
71      brk
72                                          ;should never get here
73 ErrCount DFB      0
74
75 ParmTbl DS      5

```

Listing 2 - The Embedded String Printing Module

```

1
2 *****
3 *
4 *  Embedded String Printer
5 *
6 *****
7
8      mx      %11
9      REL
10     DSK      EMBEDSTR.PRTR.L
11     LST      OFF
12
13 * Equates
14
15 HTAB    =      $24
16 OurPtr  =      $06                      ;zero page pointer
17 COUT    =      $FDED
18 CROUT   =      $FD8E                    ;generate a carriage return
19
20
21 * Start of printing module
22
23 Imprint ENT                      ;a global label
24
25      LDA      OurPtr                  ;get previous contents of $06
26      STA      PTRSAVE                  ;save it in our own data table
27      LDA      OurPtr+1

```

```

28      STA  PTRSAVE+1          ;do likewise for $07
29
30      PLA
31      STA  OurPtr             ;pull return address off stack
32      PLA
33      STA  OurPtr+1
34
35      LDX  #0                  ;move cursor flush left
36      STX  HTAB
37      JSR  CROUT              ;move down a line from last cursor
38      LDY  #0
39
40  nextchr2  INC  OurPtr         ;inc pointer to point at text
41          BNE  nextchr
42          INC  OurPtr+1        ;if it rolled, inc highbyte, too
43
44  nextchr   LDA  (OurPtr),Y     ;get character
45          BEQ  exit4          ;terminate on zero
46          JSR  COUT
47          JMP  nextchr2
48
49  exit4     LDA  OurPtr+1       ;get hbyte of return address
50          PHA                      ;push back onto stack
51          LDA  OurPtr          ;get lobyte
52          PHA                      ;and push back onto stack
53
54          LDA  PTRSAVE+1       ;restore zero page
55          STA  OurPtr+1
56          LDA  PTRSAVE
57          STA  OurPtr
58          RTS
59
60  PTRSAVE  DS    2             ;data table

```

The error handling module itself does some peculiar things. Let's pick 'em apart.

First, it scans the list of error numbers looking for a match. It increments the X register so that when a match is found it can use X as an offset into a jump table. The jump table that begins at line 89 (JMPFL) is a list of the addresses of our error handlers. There is an error handler for each error (although if you look at the handlers themselves several of them handle more than one error).

When a match is found, the routine moves the X register into the accumulator, shifts left to double it, then moves it back into X. Since the addresses in the table at JMPFL are two bytes each, the offset needs to be doubled in this fashion to point us to the correct error handling routine.

The final bit of weirdness is the manner in which I actually did the jump. Instead of moving the address to zero page and doing an indirect JMP (a buggy opcode on the 6502, by the way) it is faster to read each address directly and push it on the stack. Why the stack? Hmmm... well, it is a little bit of scullduggery, I must admit. We're going to fake out the CPU. If the address of the error handler is on top of the stack and we then execute an RTS, the CPU just returns control to the address sitting on top of the stack. Our silicon savant does not know whether we really JSR'd or not, and it doesn't care. The PHA highbyte, PHA lowbyte, and RTS combination is a quick and effective method for jumping who-knows-where. The lookup table of addresses combined with this technique makes for a very effective "option picker", as Don Lancaster called it in *The Assembly Language Cookbook for the Apple II/IIe*. (Although the book is getting a little long in the tooth - it discusses EDASM in depth - it still is an invaluable resource for 8 bit programmers. I'm sure Don himself could put a copy in your hands. Call 602/428-4073).

Speaking of the lookup table of addresses, you might notice that they all are the destination address less one byte. The reason for this is that the RTS returns control to the code living one byte past the address left on the stack.

Listing 3 - The Error Handling Module

```

1 *****
2 *
3 * Segment: Error Handler *
4 *
5 *****
6
7         mx      $11
8         REL
9         DSK      MLI.ERR.L
10        LST      OFF
11
12 * A few equates
13
14 OurPtr   =      $06                      ;zero page pointer
15
16 BELL      =      $FF3A
17 COUT      =      $FDED
18 CROUT     =      $FD8E                  ;generate a carriage return
19 Keyboard  =      $C000                  ;read a key
20 ClnStrobe =      $C010                  ;clears keyboard queue
21 PRHEX     =      $FDE3                  ;lower nibble of A as hex char
22
23 * Our lone external reference
24
25         EXT      Imprint
26
27 * The MLI and our demo module passes the error number in the accumulator
28
29 MLI_Error ENT
30         STA      error_number            ;store error_number
31         JSR      BELL
32         LDA      error_number
33
34         LDX      #28                      ;29 MLI errors
35 scan     CMP      errorlist,X
36         BEQ      matchfound
37         DEX
38         BNE      scan
39
40         BRK
41
42 matchfound
43         TXA
44         ASL
45         TAX
46         LDA      JMPFL+1,X                ;push page address onto stack
47         PHA
48         LDA      JMPFL,X
49         PHA
50
51         RTS
52
53
54 errorlist ENT
55
56         DFB      1                        ;List of MLI errors by number
57         DFB      4
58         DFB      $25
59         DFB      $27
60         DFB      $28
61         DFB      $2B
62         DFB      $2E
63         DFB      $40
64         DFB      $42
65         DFB      $43

```

```
66          DFB      $44
67          DFB      $45
68          DFB      $46
69          DFB      $47
70          DFB      $48
71          DFB      $49
72          DFB      $4A
73          DFB      $4B
74          DFB      $4C
75          DFB      $4D
76          DFB      $4E
77          DFB      $50
78          DFB      $51
79          DFB      $52
80          DFB      $53
81          DFB      $54
82          DFB      $55
83          DFB      $56
84          DFB      $57
85          DFB      $58
86
87 * table of error handler addresses  (all -1 'cuz RTS takes you one PAST
addr)
88
89 JMPFL      DA      err1-1
90          DA      err4-1
91          DA      err25-1
92          DA      err27-1
93          DA      err28-1
94          DA      err2B-1
95          DA      err2E-1
96          DA      err40-1
97          DA      err42-1
98          DA      err43-1
99          DA      err44-1
100         DA      err45-1
101         DA      err46-1
102         DA      err47-1
103         DA      err48-1
104         DA      err49-1
105         DA      err4A-1
106         DA      err4B-1
107         DA      err4C-1
108         DA      err4D-1
109         DA      err4E-1
110         DA      err50-1
111         DA      err51-1
112         DA      err52-1
113         DA      err53-1
114         DA      err55-1
115         DA      err56-1
116         DA      err57-1
117         DA      err58-1
118
119
120 * Multiple labels for the same address here because MLI errors (in hex)
121 * are not user-correctable.  These are programmer's problems!
122
123
124 err1      ;invalid MLI command/programmer error
125 err4      ;invalid parameter count/prog.error
126 err25     ;interrupt table full
127 err43     ;file not open error
128 err4A     ;incompatible version of ProDOS
```



```
129 err50                                ;file busy error
130 err55                                ;VCB table full
131 err56                                ;buffer in use
132
133     JSR     Imprint
134     ASC     "Error #: ",00
135     LDA     error_number                ;move high nibble down to low nibble
136     LSR
137     LSR
138     LSR
139     LSR
140     JSR     PRHEX
141     LDA     error_number
142     JSR     PRHEX                        ;print low nibble
143
144     JMP     DoPrompt
145
146 err27     JSR     Imprint
147     ASC     "I/O ERROR",8D,00
148     JMP     DoPrompt
149
150 err28     JSR     Imprint
151     ASC     "NO DEVICE CONNECTED",8D
152     ASC     "Check slot and drive selection.",00
153     JMP     DoPrompt
154
155 err2B     JSR     Imprint
156     ASC     "Your disk is write protected.",8D,00
157     JMP     DoPrompt
158
159 err40     JSR     Imprint                ;invalid pathname syntax
160     ASC     "INVALID PATHNAME",00
161     JMP     DoPrompt
162
163 err45                                ;two MLI errors related to not
164 err2E                                ;having a volume online
165     JSR     Imprint
166     ASC     "VOLUME NOT ONLINE",8D,00
167     JSR     vol_prompt
168     JMP     DoPrompt
169
170 err42     JSR     Imprint
171     ASC     "BUFFERS FULL",8D,00
172     JMP     DoPrompt
173
174 err44     JSR     Imprint
175     ASC     "DIRECTORY NOT FOUND",8D,00
176     JMP     DoPrompt
177
178 err46     JSR     Imprint
179     ASC     "FILE NOT FOUND",8D,00
180     JMP     DoPrompt
181
182 err47     JSR     Imprint
183     ASC     "DUPLICATE FILE NAME",8D,00
184     JMP     DoPrompt
185
186 err48     JSR     Imprint
187     ASC     "DISK FULL",8D,00
188     JMP     DoPrompt
189
190 err49     JSR     Imprint
191     ASC     "DIRECTORY FULL",8D,00
192     JMP     DoPrompt
```

```

193
194 err4B    JSR    Imprint
195          ASC    "FILETYPE ERROR",8D,00
196          JMP    DoPrompt
197
198 err4C    JSR    Imprint
199          ASC    "OUT OF DATA",8D,00
200          JMP    DoPrompt
201
202 err4D    JSR    Imprint
203          ASC    "RANGE ERROR",8D,00
204          JMP    DoPrompt
205
206 err4E    JSR    Imprint
207          ASC    "FILE LOCKED",8D,00
208          JMP    DoPrompt
209
210 err51    JSR    Imprint
211          ASC    "THE DIRECTORY MAY BE DAMAGED",8D,00
212          JMP    DoPrompt
213
214 err52    JSR    Imprint
215          ASC    "NOT A PRODOS DISK",8D,00
216          JMP    DoPrompt
217
218 err53    JSR    Imprint
219          ASC    "INVALID PARAMETER",8D,00
220          JMP    DoPrompt
221
222 err57    JSR    Imprint
223          ASC    "DUPLICATE VOLUMES ONLINE",8D,00
224          JMP    DoPrompt
225
226 err58    JSR    Imprint
227          ASC    "The volume bitmap may be damaged!",8D,00
228          JMP    DoPrompt
229
230
231
232 get_response
233          JSR    CROUT
234          LDA    #" "                                ;print cursor
235          JSR    COUT
236          STA    ClrStrobe
237 rdkbd     LDA    Keyboard
238          BPL    rdkbd
239          CMP    #$8D                                ;RETURN?
240          BEQ    exit3
241          CMP    #155                                ;escape?
242          BNE    rdkbd
243
244 exit3     JSR    CROUT
245          RTS
246
247
248 DoPrompt
249          JSR    Imprint
250          ASC    "Press RETURN to try again,",8D
251          ASC    "ESCAPE to abort...",00
252          JSR    get_response
253          RTS
254
255 vol_prompt
256          JSR    Imprint

```

```

257          ASC    "Please insert: ",8D,00
258
259 * This section requires a volume name (which is potentially kept in various
260 * places). For this demo I've hardcoded a fake path at Pathname. Depending
261 * on your application, you might want to do a GET_PREFIX and display that.
262
263          LDA     #<Pathname                ;put location of path into zero page
264          STA     OurPtr
265          LDA     #>Pathname
266          STA     OurPtr+1
267
268          LDX     Pathname                    ;length of string
269          LDY     #1                          ;offset to skip length byte
270 :loop     LDA     (OurPtr),Y
271          ORA     #$80                        ;clean up display
272          JSR     COUT
273          INY
274          DEX
275          BEQ     history
276          JMP     :loop
277
278 history   JSR     CROUT
279          RTS
280
281 error_number DFB 0
282
283 Pathname STR    "/THIS.IS.A.TEST"

```

Listing 4 - Linker Names File Creator (Merlin 8 only)

```

1 *****
2 *
3 *  Names File Creator for Merlin 8 Linker
4 *
5 *****
6
7
8          DSK     MLI.NAMES
9          STR     "DemoModule.L"
10         STR     "Embedstr.prtr.L"
11         STR     "MLI.ERR.L"
12         BRK

```

Listing 5 - Linker Command File (Merlin 16 only)

```

1 *****
2 *
3 *
4 *  Linker Command File for P8 MLI Error Routines
5 *  (Merlin 16 only)
6 *
7 *****
8
9          org     $2000                      ;let's create a SYS file
10         typ     $FF
11
12         lkv     $00                          ;specify absolute linker (P8)
13
14         asm     mli.err.link.s                ;change to your names for each
15         asm     str.printer.s                ;if you rename them!
16         asm     mli.err.demo1.s
17

```

```

18      lnk    demomodule.l
19      lnk    embedstr.prtr.l
20      lnk    mli.err.l
21
22      sav    MLI.ERR.DEMO

```

I had Merlin 8/16 and then got the update to Merlin 16+. The additional documentation I received did not point out that you could link 8 bit files with no hassle using the 16+ linker. Through a little experimentation, I discovered that the LKV \$00 pseudo op still invokes the absolute linker, so your eight bit code links like a charm even in Merlin 16+. And at the risk of provoking the ire of all you IIC and IIC fans, I am compelled to add that the IIGS and Merlin 16+ is an absolutely *incredible* 8 bit programming environment. The command files of the linker are flexible, powerful, and easy to use, and the linker itself is like lightning. All of the files in this program linked and saved to disk in 3 seconds to my Applied Ingenuity Inner Drive.

For some perverse reason it is tempting, when starting a new project, to write the entire thing from scratch. Hopefully our example of re-usable, linkable files will help at least some of you to discover the speed and power inherent within a more modular style.

== Ross ==

Magic Text : Using USR

More Merlin Magic From Jerry K

By Jerry Kindall, Contributing Editor

MagicText is a USR function for Merlin 8/16. It was designed for maximum flexibility in entering TXT strings. In fact, MagicText can replace all of Merlin's text opcodes, except for STR (and that's only because I couldn't fit the code to handle a leading length byte into page 3 of RAM).

To install MagicText, you simply press D (for Disk Command) at Merlin's main menu, then type BRUN MAGICTEXT. Once you've done that, MagicText will be installed and ready to use. (You can also automatically run MagicText when you run Merlin by putting its pathname into Merlin's startup buffer, but then Merlin wouldn't load the full screen editor automatically.)

Using MagicText

MagicText is activated by a USR psuedo-op in your source code. (If you use Merlin 16, use USR0 instead of USR.) A typical MagicText statement might look like this:

```
greeting usr 'Hi there!' ;greeting string
```

That's a simplistic example, of course, and it doesn't show you the flexibility of MagicText at all. However, notice that, just as with any other Merlin psuedo-op, you have an optional label, the opcode, the operand, and an optional comment.

MagicText will allow you to use any character at all (except the tilde character, ~) as a delimiter for the string, but I suggest the use of the apostrophe or quote. With MagicText, there's no reason to ever need more than one delimiter.

MagicText works its magic by means of the tilde character. The tilde has special meaning in MagicText strings. For example, if you put ~A in a MagicText string, MagicText will insert a control-A character into the string. (In fact, any character in the ASCII range 64-95, which includes the uppercase letters and the symbols @, [, \,], ^, and _, will generate a control character when preceded by a tilde.)

Here's an example, which contains two bell characters embedded in the text:

```
usr  ~~GAre you awake?~G~      ;awaken user
```

If you follow the tilde with another tilde, MagicText will put one tilde character into the object code. If you follow the tilde with a quote mark or an apostrophe, MagicText will insert those characters as well, even if you're using one of them as a delimiter. Here's an example:

```
usr  "Joe said, ~"I am going to the store.~"
```

If you follow the tilde with a dollar sign, MagicText will interpret the two characters after the dollar sign as a hex byte. Here's an example of using this feature to terminate a string with a carriage return and a zero byte:

```
usr      "Main menu - Please make a  
selection~$8D~$00"
```

MagicText also recognizes a few lower-case letters after the tilde, as flags to change modes. Remember, if you use upper-case letters, MagicText will consider the letter a control-character. (Note: ~l is a lower-case letter L, not the numeral one.)

```
~l: Switch to low-ASCII (high bit clr) chars  
~h: Switch to hi-ASCII (high bit set) chars  
~i: Switch to inverse text  
~f: Switch to flashing text  
~m: Switch to MouseText  
~n: Switch back to normal text (high-ASCII)
```

MagicText uses the ~l and ~h flags to select high or low ASCII text, instead of looking at the delimiter. Text is always assumed high ASCII unless you use the ~l flag to specify low ASCII. (MagicText passes all characters except hex bytes through the high/low ASCII flag, including control characters and the bytes generated by ~~, ~', and ~".)

The ~i, ~f, and ~m flags cause MagicText to manipulate the ASCII codes of your text to produce the desired types of characters. Inverse text works properly in 80-column mode, with both upper and lower case (in 40-column mode, lower case inverse text is displayed as flashing punctuation and numerals). Flashing text does not support lower-case. MouseText expects you to specify an ASCII code in the range of 64-95 (the letters and symbols @, [, \,], ^, and _).

The display flags ~i, ~f, and ~m are useful mostly for applications that will be storing characters directly to screen memory, or using only the 40-column output routines. The 80-column firmware will ignore some of these ASCII codes or treat them as control characters (in particular, the uppercase inverse letters).

The ~n flag is actually the same as ~h and sets high-ASCII normal characters. The ~l flag will also turn off ~i, ~f, or ~m, and switch to low-ASCII characters. Here's an example which generates the ASCII codes for a small mousetext box:

```
usr  "~mZ\\\\^"
```

Here's another example with an inversed word:

```
usr  "It's time to~i PARTY ~n"
```

How Does It Work?

If you're not familiar with Merlin's USR opcode, you should check out pages 124 and 125 in the Merlin 8/16 manual. (That information probably moved around somewhat when Merlin 16+ was released. Check the index if you don't find it on pages 124-125.)

MagicText starts out by hooking itself up to Merlin's USR vector (lines 80-90). Notice that the code which does this actually resides in the input buffer, but since that code won't be needed again, it's OK to put it in such an unstable memory location. The actual USR routine starts at address \$300.

The first thing MagicText does when it gets control is determine the delimiter being used and to initialize a few flags (lines 92-103). Then it falls into the main processing loop (lines 105-135), which processes each character in the operand. If a tilde is found, the tilde routine (lines 159-187) gets control, and examines the character after the tilde to figure out what to do. If a tilde is not found, the current mode (lo/hi ASCII, inverse/flash/mousetext) is checked and the character is adjusted accordingly before being placed into the object code.

The tilde routine checks for ~, ', and " characters, and if it finds them following a tilde, places them into the object code via PROC (line 112). Next it checks for h, l, i, f, m, and n; if they are found, the appropriate mode is set. If a dollar sign is found, the hex byte routine is activated. If none of these characters are found, the character is converted to a control character and put into the object code (lines 183-186).

The hex byte routine (192-200) calls the hex digit routine (206-219) twice, once for each nibble, then combines the two nibbles into a byte and puts them into the object code.

The code is a little bit tricky in places because of my desire to fit it into page 3 of RAM, but is otherwise fairly straightforward. It's a good example of how to write a USR routine for Merlin.

I've found MagicText quite useful in my programming. I hope you find it useful in yours. Enjoy!

Listing 1 : MagicText Assembly Listing

```

1  *****
2  *
3  *           M a g i c T e x t           *
4  *
5  *           A Merlin 8/16 USR Routine    *
6  *           by Jerry E. Kindall         *
7  *           August 1989                 *
8  *
9  *           Public Domain               *
10 *
11 *****
12
13
14
15 * MagicText is a replacement for all of Merlin's
16 * various text-generation psuedo-ops. It allows
17 * you to switch between high ASCII, low ASCII,
18 * inverse, normal, flashing, and mousetext, and
19 * to insert control characters and hex bytes,
20 * all in the same source statement. The only
21 * thing that MagicText can't do is produce a
22 * leading length byte - you'll still have to use
23 * STR for that.
24
25
26
27 * Syntax:
28 * USR 'text' ;comment
29 *
30 * The apostrophe is a delimiter and can be any
31 * character except ~, and it must be matched
32 * by another such character. Apostrophe or quote
33 * recommended. An optional comment may follow.
34 *
35 * If a tilde (~) is encountered in the text, the
36 * tilde and the character that follows it are
37 * treated specially. The following characters
38 * are valid after a tilde (all letters MUST be
39 * lower case):
40 *
41 *
42 * h: switch to high-ASCII characters
43 * n: switch to normal (high-ASCII) characters
44 * i: switch to inverse characters
45 * f: switch to flashing characters
46 * m: switch to mousetext characters
47 * ~: insert a tilde (ie, ~~ = one tilde)
48 * ': insert an apostrophe (ie, ~' = one apost)
49 * ": insert a quote (ie, ~" gives one quote)
50 * $: the next two characters are a hex byte;
51 * ~$0D inserts the hex value 0D
52 *
53 * Any other characters are considered control
54 * chars: ~A inserts a control-A, etc
55
56
57
58             org         $2F0

```

```

59
60
61
62 * Internal Merlin Entry Points:
63 * See Merlin 8/16 Manual, pp 124-125
64
65         opndlen = $BB           ;length of operand
66         worksp  = $280          ;operand work buffer
67         usrvect = $B6DA         ;USR routine vector
68         putbyte = $E5F6         ;put a byte into object
69
70
71
72 * Zero page locations used by this routine:
73 * Allocated by Merlin as temporary storage
74
75         dlimit  = $60           ;string delimiter
76         mode     = $61           ;ASCII mode
77         hold     = $62           ;temporary storage
78
79
80
81 * Connect the USR routine to Merlin
82
02F0: A9 4C 83  setup      lda     #$4C
02F2: 8D DA B6 84          sta     usrvect
02F5: A9 00 85          lda     #usrop
02F7: 8D DB B6 86          sta     usrvect+1
02FA: A9 03 87          lda     #/usrop
02FC: 8D DC B6 88          sta     usrvect+2
02FF: 60 89          rts
90
91
92
93 * USR psuedo-op entry point
94 *
95 * On entry from Merlin: A = 0, Y = 0, carry = 1
96
0300: A9 80 97  usrop      lda     #$80
0302: 85 61 98          sta     mode           ;high ASCII (normal) mode
0304: 85 60 99          sta     dlimit        ;no realdelimiter
0306: 20 B9 03 100        jsr     get           ;get firstchar of opernd
0309: F0 38 101        beq     done          ;we're at end of line
030B: 85 60 102        sta     dlimit        ;we have the delimiter
103
104
105
106 * Main text processing loop
107
030D: 20 B9 03 108  loop      jsr     get           ;get next char of operand
0310: F0 31 109        beq     done          ;at end, we're done
0312: C9 7E 110        cmp     #'~'         ;is it a command?
0314: F0 41 111        beq     tilde        ;yes, go do it
0316: A6 61 112  proc      ldx     mode        ;what mode we in?
0318: F0 23 113        beq     lo           ;low ASCII mode
031A: 30 19 114        bmi     hi           ;high ASCII mode
031C: E0 02 115        cpx     #2
031E: 90 19 116        blt     mst          ;1 = mousetext mode
0320: F0 06 117        beq     inv          ;2 = inverse mode
0322: 29 3F 118  fls      and     #$3F        ;else flashing mode
0324: 09 40 119        ora     #$40
0326: D0 15 120        bne     lo           ;put the character
0328: C9 40 121  inv      cmp     #$40        ;less than 64, OK already
032A: 90 11 122        blt     lo

```

```

032C: C9 60      123      cmp    #$50      ;greater than 96, it's OK
032E: B0 0D      124      bge    lo
0330: 29 3F      125      and    #00111111 ;convert to 0-32
0332: 4C 3D 03    126      jmp    lo        ;and put the char
0335: 09 80      127      hi      ora    #10000000 ;set hi bit of char
0337: D0 04      128      bne    lo        ;and put it
0339: 29 3F      129      mst     and    #00111111 ;convert to 0-32...
033B: 09 40      130      ora    #01000000 ;convert to 64-95
033D: 20 F6 E5    131      lo      jsr    putbyte ;put the character
0340: 4C 0D 03    132      jmp    loop     ;and go back to the top
                                133
0343: 60          134      done    rts          ;we're all done!
                                135
                                136
                                137
                                138      * Set the various text modes
                                139
0344: A9 03      140      setfls   lda    #$03      ;mode = 3 (flash)
0346: 2C          141      hex     2C          ;fake BIT to skip next instr
                                142
0347: A9 80      143      sethi    lda    #$80      ;mode = $80 (norm/hi)
0349: 2C          144      hex     2C
                                145
034A: A9 00      146      setlo    lda    #$00      ;mode = 0 (lo ASCII)
034C: 2C          147      hex     2C
                                148
034D: A9 01      149      setmst   lda    #$01      ;mode = 1 (mousetext)
034F: 2C          150      hex     2C
                                151
0350: A9 02      152      setinv    lda    #$02      ;mode = 2 (inverse)
0352: 85 61      153      sta     mode     ;set it
0354: 4C 0D 03    154      jmp     loop     ;back to the top
                                155
                                156
                                157
                                158      * Handle tilde commands
                                159
0357: 20 B9 03    160      tilde    jsr     get      ;get char after tilde
035A: C9 7E      161      cmp     #'~'      ;it's a tilde, do it
035C: F0 B8      162      beq     proc
035E: C9 27      163      cmp     #$27      ;it's an apost, do it
0360: F0 B4      164      beq     proc
0362: C9 22      165      cmp     #'"'      ;quote, do it
0364: F0 B0      166      beq     proc
0366: 20 BD 03    167      jsr     check     ;is it a delimiter?
0369: F0 D8      168      beq     done     ;it is, exit
036B: C9 24      169      cmp     #'$'      ;$ = hex mode
036D: F0 20      170      beq     hex
036F: C9 68      171      cmp     #'h'      ;set high ASCII
0371: F0 D4      172      beq     sethi
0373: C9 6C      173      cmp     #'l'      ;set lo ASCII
0375: F0 D3      174      beq     setlo
0377: C9 69      175      cmp     #'i'      ;set inverse
0379: F0 D5      176      beq     setinv
037B: C9 6E      177      cmp     #'n'      ;set normal (high)
037D: F0 C8      178      beq     sethi
037F: C9 6D      179      cmp     #'m'      ;set mousetext
0381: F0 CA      180      beq     setmst
0383: C9 66      181      cmp     #'f'      ;set flashing
0385: F0 BD      182      beq     setfls
0387: 29 1F      183      and    #00011111 ;it's a ctrl-char
0389: A6 61      184      ldx     mode
038B: F0 B0      185      beq     lo        ;if low ASCII on, set low
038D: D0 A6      186      bne     hi        ;otherwise, set high

```



```

187
188
189
190 * Handle hex bytes.
191
038F: 20 A0 03 192 hex      jsr      dig      ;get one hex digit
0392: 0A      193          asl      ;mult by 16
0393: 0A      194          asl
0394: 0A      195          asl
0395: 0A      196          asl
0396: 85 62    197          sta      hold    ;hold it
0398: 20 A0 03 198          jsr      dig      ;get next digit
039B: 05 62    199          ora      hold    ;combine with hold
039D: 4C 3D 03 200          jmp      lo      ;and store it
201
202
203
204 * Get a hex digit from operand
205
03A0: 20 B9 03 206 dig      jsr      get      ;get a char
03A3: D0 04    207          bne      valid    ;it's AOK
03A5: 88      208          dey      ;at EOL, return 0
03A6: A9 00    209          lda      #0
03A8: 60      210          rts          ;back
03A9: C9 60    211 valid    cmp      #$60    ;lower case?
03AB: 90 02    212          blt      conv     ;no
03AD: E9 20    213          sbc      #$20    ;yes, fix to upper
03AF: 38      214 conv     sec          ;convert ASCII to hex
03B0: E9 30    215          sbc      #'0'
03B2: C9 0A    216          cmp      #10     ;it's a number
03B4: 90 02    217          blt      back
03B6: E9 07    218          sbc      #7      ;must be a letter; fix it
03B8: 60      219 back     rts
220
221
222
223 * Get one character from operand
224
03B9: B9 80 02 225 get      lda      worksp,y ;get char from workspace
03BC: C8      226          iny          ;and point to next
03BD: C5 60    227 check    cmp      dlimit    ;hit delimiter?
03BF: F0 02    228          beq      rts      ;yep
03C1: C4 BB    229          cpy      opndlen  ;at end of line?
03C3: 60      230 rts      rts

```

Unleashing TextEdit

By Jay Jennings

Probably the most exciting new tool included with System Disk 5.0 is the TextEdit toolset. The old LineEdit toolset allows a user to enter and edit a single line of text in a program. TextEdit allows the user to enter multiple lines. In fact, TextEdit can be thought of as a full featured word processor. By full featured, I mean it supports multiple fonts, styles, and colors in the text, full editing according to the Human Interface Guidelines, and can support a document of virtually unlimited size...all this with

one toolset!

The purpose of this article is to show you how to create a little text editor with just a few lines of code. We won't go into different fonts, styles, and colors, however. That would take more pages than Ross will let me have. But we will include the load and most of the save code.

Our program starts all needed tools, allocates a 64K

buffer for our text, creates a window, installs a TextEdit control, and then heads for the Event loop. There's no menu bar in this program. To quit the program, click on the close box of the window.

The program currently doesn't save the text. You could do that by adding the TEGetText call (explained in the article) and then writing the data to disk. This program also lacks any error checking. I left that out because of space, but you should check for errors after every tool call.

Let's skip the descriptions of the normal stuff like opening windows and go straight for the throat of the TextEdit control. Then we'll back up and see how to install it in a window using NewControl2.

The first parameter in the template is a parameter count. You can have as few as 7 parameters in the template or as many as 23. This depends on how many of Apple's defaults you want to accept. For our purposes, all we need are 18 parameters.

```
dw 18
```

The second parameter is the ID of our control. This needs to be unique for the window in which the TextEdit control resides. Just pick your favorite number. Notice that in the next line I use the "dl" pseudo-op. This is a macro that takes the place of the "adrl" pseudo-op just because define long makes more sense when defining a long number than adrl does. Right? *(Editor: I think so. I've always felt funny using ADRL - ADdResS Long- when defining flags or other non-address sorts of things.)*

```
dl 7
```

Parameter number three is four word values that specify the boundary rectangle for the TextEdit control.

```
dw 5,5,170,610
```

The fourth parameter is the actual value that indicates you're implementing a TextEdit control.

```
dl $85000000
```

The next two parameters are flags that specify how the TextEdit control will act while being used. The first of the two flag words must be set to zero. The second is a little more flexible, but 9 times out of 10 you'll need to set it exactly as I show it here.

```
dw 0
dw $0111_0100_0000_0000
```

Parameter number seven is a long space that is left blank. It's for our use so we can put anything we want in there. Well, anything that's not over four bytes long, anyway.

Now we get to the "grand-daddy" parameter... number eight. There are a zillion bits that mean a zillion different things (give or take a few). I'm just going to go through a few of the more important ones. Those I don't mention, just leave them as is until you latch onto the docs for the TextEdit toolset in the Apple IIGS Toolbox Reference, volume 3.4. *(Editor: or until a future SApp article)*

```
Bit 28  0 = word wrap the text
         1 = break at CR only
Bit 27  0 = scrolling permitted
         1 = no manual or autoscrolling
Bit 26  0 = editing permitted
         1 = no editing allowed
Bit 24  0 = tab inserted in document
         1 = tab to next cntrl in window
Bit 23  0 = no rect around TE control
         1 = draw rect around TE control
Bit 20  0 = user can select text
         1 = user cannot select text
```

Here's the way the parameter looks for a "generic" kind of TextEdit control...

```
dl $0110_1010_1010_0000_0000_0000_0000_0000
```

Parameter number nine (actually four words) describes the amount of white space to leave between the boundary rectangle and the text itself. The default values of 2, 6, 2, and 4 (top, left, bottom, right) can be specified by using \$FFFF for each parameter.

```
dw $FFFF,$FFFF,$FFFF,$FFFF
```

Parameters ten and eleven concern the vertical scroll bar. Set them both to zero if you don't want a vertical scroll bar. If you'd like a scroll bar without any hassles, set parameter ten to \$FFFF (or -1) and parameter eleven to zero. This will give you a scroll bar that scrolls 9 pixels at a time.

```
dl $FFFF
dw 0
```

The horizontal scroll bar is handled by parameters twelve and thirteen and are dealt with just like the vertical scroll bar was. Well, they will be, but horizontal scrolling isn't implemented yet. For now, they MUST be set to zero or bad things will happen to you and your computer.

```
dl 0
dw 0
```

The next four parameters (fourteen through seventeen) are complicated enough that you'll need the manual to make good use of them. Just leave them as they are for the purposes of our demo code.

```
dl 0 ;ref to style information
dw 0 ;textDescriptor
dl 0 ;ref to initial text
dl 0 ;length of initial text
```

The last parameter we'll deal with sets the maximum number of characters that we want our control to allow. Since our program sets up a 64K buffer for text, we'll specify that as the maximum size.

```
dl 65535
```

Phew! The TextEdit template is done. Now we'll dive into the other two calls that are used with TextEdit quite a lot. The first, TSESetText, grabs text from a buffer in memory and places it into the TextEdit document. The other, TEGetText, grabs the text from the TextEdit record and places it in a buffer. Then you'd be ready to save it to disk, transmit it over the modem, or whatever else you desired.

There are six parameters that need to be pushed on the stack for TSESetText. The first defines the format of the next parameter. Bits 3-4 show the next parameter is a pointer. Bits 0-2 specify that we're after an unformatted block of text. We're going to take the easy way out and use all pointers in our example (we could use handles or resource IDs if we wanted to get sneaky). That means the second parameter is a pointer to the text that will be inserted in the TextEdit document. The third parameter specifies the number of characters in the text buffer. The next two parameters should be set to zero as they're for style information and we won't be getting into that at this time. The last parameter is the handle to the TEREcord in memory. But, we don't even have to worry about that too much because if we put a zero in that parameter it will default to the active record. Here's what the parameter list looks like for our program...

```
PushWord #00101 ;textDescriptor
PushLong TextBuffer ;textRef
PushLong TextLength ;textLength
PushWord #0 ;styleDescriptor

PushLong #0 ;styleRef
PushLong #0 ;teHandle
_TSESetText ;make the call!
```

The format for TEGetText is very similar. Since the call is going to return a result, we have to push space on the stack first. And instead of pointing to a block of text in memory, we point to a block of space that the text will end up in after the call.

```
PushLong #0 ;space for result
PushWord #00101 ;bufferDescriptor
PushLong TextBuffer ;bufferRef
PushLong #65535 ;bufferLength
PushWord #0 ;styleDescriptor
PushLong #0 ;styleRef
PushLong #0 ;teHandle
_TEGetText ;yank out data
PullLong TotalLength ;length of all
text in record
```

In order for the TextEdit control to become active it has to be installed in our window. We use the NewControl2 call and install it just like any other control, like a button, checkbox, or edit line. The use (and abuse) of NewControl2 is a subject for the future, so for now, just stare very hard at that part of the source code and absorb the subtle intricacies through osmosis. Okay, I'll explain the parameters here very briefly.

You push a long space on the stack first. The call returns a handle to the control although we don't do anything with that value in our program.

The second parameter is the pointer to the window you want to install the control in. That value is the one returned in the NewWindow call made earlier.

The third parameter is a reference for the fourth, and last parameter. By pushing a zero we're saying that the next parameter is a pointer to the template of a single control. By using different values for the third parameter we can specify that the last parameter will be a handle, pointer, or resource ID of a single template or table of templates. NewControl2 is a very handy call. It's made window-type programming very quick and easy (until you get to line edit controls...which is a subject for a future article). Here's what the NewControl2 call should look like...

```
PushLong #0 ;space for result
PushLong WindowPtr ;ptr wndw cntrl
PushWord #0 ;ref descriptor
PushLong #Template ;address of cntrl template
_NewControl2
PullLong TEHandle ;retrieve cntrl hnd1
```

That's it! You know everything needed to become a TextEdit guru. Well, you know enough to get started on it, anyway. Look over the source code and follow the logic to see what's happening.

Editor: You'll notice that Jay does a JSR StartUp and JSR Shutdown - those are calls to routines virtually identical to the Generic Start II we ran last time. They can (and probably should) be put into reusable, linkable files. The only time they'd need to be changed is when your current application needs more tools than are included in those generic routines.

Incidentally, we had two reports of difficulties with

Generic Start II, but had no luck tracking down the bug. We did find that the tools requested did not equal the tools listed in the StartStopRec, but that was not actually a fatal error. Jay and I both are using his code with no trouble, and the other person I sent a copy to has reported no problems either.

That doesn't mean that anyone is crazy, of course, it just means that we couldn't replicate the problem (no response from TaskMaster).

```

1          1st    off
2 *=====
3 * Mini word processor for The Sourceror's Apprentice
4 * Another Mohawk Man Creation
5 * Copyright 1989 - PunkWare
6 *=====
7          xc
8          xc
9          mx      $00
10         cas     in
11         rel
12         use      mwp.macs
13         put      1/tool.equates/e16.window
14         put      1/tool.equates/e16.memory
15         put      1/tool.equates/e16.gsos
16 *-----
17         do      0
18 dl      mac      ;a new macro
19         adr1    ]1
20         eom
21         fin
22 *-----
23         phk
24         plb      ;set data and program bank the same
25         jsr      StartUp ;load and start the tools
26         jsr      MemAlloc ;grab a 64K chunk for data
27         jsr      MakeWindow ;a window for TextEdit to live in
28         jsr      WakeTextEdit ;...and make it active
29         jsr      GetFile ;choose a file to load
30         bcs      :NoFile ;if cancel was clicked, branch
31         jsr      SetText ;put the text in the window
32 :NoFile
33         _InitCursor
34         jsr      EventLoop ;go do that loop thing
35         jmp      ShutDown ;and exit the program
36 *-----
37 GetFile
38         ~SGetFile2 #120;#40;#0;#Prompt1;#0;#0;#ReplyRec
39         lda      ReplyRec ;see what was clicked
40         bne      :Load ;if file picked, go load it
41         sec
42         rts
43 :Load
44         iGSOS _Open;OpenParms;1

```

```

45      lda    OpenRefNum
46      sta    ReadRefNum
47      sta    CloseRefNum
48      MoveLong OpenEOF;ReadRequest      ;move the length of file
49      MoveLong BufferPointer;ReadBuffer  ;move the buffer address
50      iGSOS _Read;ReadParms;1
51      iGSOS _Close;CloseParms;1
52      MoveLong BufferPointer;50          ;move address to direct page
53      ldy    OpenEOF                    ;get length of file if < 64K
54      sep    $20                        ;go to 8 bit accumulator
55 ]loop  lda    [50],y                    ;grab a character
56      and    #$7F                        ;strip off the hight bit
57      sta    [50],y                      ;and resave it
58      dey                                ;point to the previous character
59      bpl    ]loop                       ;if not -1, keep looping
60      rep    $20                        ;back to 16 bit accumulator
61      clc
62      rts
63 *-----
64 SetText
65      ~TESetText %101;BufferPointer;OpenEOF;#0;#0;#0
66      rts
67 *-----
68 Startup
69      _TLStartup      ;tool locator first
70      ~MMStartup #0      ;start the mem manager
71      PullWord ProgID
72      _MTStartup      ;misc tools manager
73      ~StartUpTools ProgID;#0;#StartStopRec
74      PullLong SSRec
75      rts
76 *-----
77 ShutDown
78      ~ShutDownTools #0;SSRec      ;kill everything we started
79      _MTShutDown
80      ~MMShutDown ProgID
81      _TLShutDown
82      iGSOS _Quit;:QParms;1
83 :QParms  ds    2
84          ds    4
85 *-----
86 MemAlloc
87      ~NewHandle #63999;ProgID;#attrLocked;#0
88      PullLong BufferHandle
89      Deref BufferHandle;BufferPointer
90      rts
91 *-----
92 MakeWindow
93      ~NewWindow #WindowTemplate
94      PullLong WindowPtr            ;grab and save the pointer
95      rts
96 *-----
97 WakeTextEdit
98      ~NewControl2 WindowPtr;#0;#TETemplate
99      PullLong TEHandle              ;save the TextEdit handle
100     rts
101 *-----

```

```
102 EventLoop
103     ~TaskMaster #$FFFF;#EventRec
104     pla                                ;get the event code
105     beq     EventLoop                 ;if nothing, keep looping
106
107     cmp     #wInGoAway                ;if window close box was clicked...
108     bne     EventLoop                 ;...then we're done
109     rts
110 *-----
111 ContentDraw
112     ~DrawControls WindowPtr
113     rtl
114 *-----
115 ReplyRec
116     dw      0                        ;good or bad?
117     dw      0                        ;type
118     dl      0                        ;auxtype
119     dw      0                        ;type of reference
120     adr1    FileName                 ;filename reference
121     dw      0                        ;type of reference
122     adr1    PathName                 ;pathname reference
123 FileName dw      19
124         ds      17
125 PathName dw      68
126         ds      64
127
128 Prompt1  str      'Choose a file to load:'
129 Prompt2  str      'Save file as:'
130 DefaultName dw 10
131         str1     'Sample'
132 OpenParms dw      12
133 OpenRefNum ds      2                ;ref number of newly opened file
134         adr1     FileName+2
135         dw      0
136         dw      0
137         ds      2
138         ds      2                    ;filetype
139         ds      4
140         ds      2
141         ds      8
142         ds      8
143         ds      4
144 OpenEOF  ds      4                ;length of newly opened file
145
146 ReadParms dw      4
147 ReadRefNum ds      2
148 ReadBuffer ds      4
149 ReadRequest ds      4
150 ReadTransfer ds      4
151
152 CloseParms dw      1
153 CloseRefNum ds      2
154
155 SSRec     ds      4
156 ProgID    ds      2
157
158 StartStopRec
159         dw      0
160         dw      $80                ; 640 mode
```

```

161      dw      0
162      adr1    0      ; dpage handle
163      dw      17     ; number of tools
164
165      dw      $1e,$0100      ;Resource
166      dw      $04,$0300      ;quickdraw
167      dw      18,$0201      ;qdaux
168      dw      $06,$0300      ;event
169      dw      27,$0300      ;font
170      dw      14,$0300      ;window
171      dw      16,$0300      ;control
172      dw      15,$0300      ;menu
173      dw      $1c,$0300      ;list
174      dw      20,$0300      ;lined
175      dw      21,$0101      ;dialog
176      dw      22,$0101      ;scrap
177      dw      5,$0101       ; desk
178      dw      23,$0101      ;file
179      dw      $13,$0200      ;print manager
180      dw      $22,$0100      ;TextEdit
181      dw      $8,$0101
182
183 BufferHandle ds 4
184 BufferPointer ds 4
185 WindowPtr ds 4
186
187 EventRec
188 eWhat      ds      2      ;event code
189 eMessage   ds      4      ;event result
190 eWhen      ds      4      ;ticks since startup
191 eWhere     ds      4      ;global mouse location
192 eModifiers ds      2      ;status of modifier keys
193 TaskData   ds      4
194 TaskMask   adr1    $001f5fff
195           adr1    0
196           adr1    0
197           dw      0
198           adr1    0
199           adr1    0
200           adr1    0
201           adr1    0
202
203 WindowTemplate
204           dw      :end-WindowTemplate      ; parm list length
205           dw      %1100000011101001      ; frame bits
206           adr1    windowtitle      ; pointer to title
207           ds      4      ; refcon
208           dw      11,0,199,630      ; zoomed rectangle
209           adr1    0      ; color table pointer
210           dw      0      ; vert offset of content
211           dw      0      ; horiz offset of content
212           dw      0      ; data area height
213           dw      0      ; data area width
214           dw      0      ; max grow height
215           dw      0      ; max grow width
216           dw      0      ; vert. arrow scroll amount
217           dw      0      ; horiz arrow scroll amount
218           dw      0      ; vert. page amount
219           dw      0      ; horiz page amount

```

```

220      adr1  0          ; info bar ref con
221      dw    0          ; info bar height
222      adr1  0          ; window procedure
223      adr1  0          ; info bar draw routine
224      adr1  ContentDraw      ; window content draw rtn
225      dw    26,2,198,637      ; starting position rect
226      adr1  -1          ; window plane, -1 is front
227      adr1  0          ; memory for window,
228 :end
229 WindowTitle str ' TextEdit Example '
230
231 TEHandle ds      4
232
233 TETemplate
234      dw    18          ;number of parameters
235      adr1  900          ;control ID
236      dw    5,5,170,610      ;boundary rectangle
237      adr1  $85000000      ;editTextControl
238      dw    0          ; flags
239      dw    %0111_1100_0000_0000 ;more flags
240      ds    4          ;refcon
241      adr1  %0110_0010_1010_0000_0000_0000_0000_0000 ;
242      dw    $ffff,$ffff,$ffff,$ffff ; indent rect defs, standards
243      dl    -1          ;make a default vert scroll bar
244      dw    0          ;vert scroll amount - 0 = default
245      dl    0          ;start with no horiz scroll bar
246      dw    0          ;horz scroll amount
247      dl    0          ;ref to style information
248      dw    0          ;textDescriptor
249      dl    0          ;reference to initial text
250      dl    0          ;length of initial text
251      dl    65535      ;max num of chars allowed
252
253 *=====
254
255      sav    mwp.1

```

The Sourceror's Apprentice

Copyright (C) 1989 by Ariel Publishing Box 398 Pateros, WA 98846 (509) 923-2249 GENie: R.W.LAMBERT
 All Rights Reserved Apple, Apple II, IIgs, BASIC.SYSTEM, and ProDOS are registered trademarks of Apple Computers, Inc.

Subscription prices in US dollars (Canada and Mexico add \$5, non-North American orders add \$18 per year)
 1 year...\$29.95 2 years...\$56 Back issues are \$3 each (non-USA add \$2) There is a quarterly source code diskette available for \$25 per
 year (Canada and Mexico add \$5, non-North American orders add \$15)

WARRANTY AND LIMITATION OF LIABILITY: I warrant that the information in The Apprentice is correct and somewhat useful to somebody
 somewhere. Any subscriber may ask for a full refund of their last subscription payment at any time. At no time shall I or my contributors be
 held liable for any incidental or consequential damages in excess of the fees paid by a subscriber.

We here at Ariel Publishing freely admit our shortcomings, but nevertheless strive to bring glory to the Lord Jesus Christ.
 (Hi Nate!)